Application of Reconfigurable Computing to Acoustic Sensors

October 2000

Andree Filipov
U.S. Army Research Laboratory
Adelphi, MD 20783

Mark Falco
Sanders
Hudson, NH 03051

Abstract

A hybrid-computing architecture consisting of a general purpose digital signal processor (DSP) closely coupled to a field-programmable gate array serving as a reconfigurable processor has been demonstrated as having great utility to collect, process, and report data from a sensor node containing acoustic and possibly other sensors. The computing architecture for microsensors (CAμS) was developed jointly by ARL and Sanders under the auspices of the Advanced Sensors FedLab Program. CAμS was used to replace an existing PCI-based computer chassis in a data collection system resulting in a ten thousand-fold improvement in size × weight × power product. The processing load is split between the general purpose DSP and the reconfigurable processor to achieve these improved results. Although in this incarnation, reconfiguration on the fly was not implemented, this paper will discuss situations where this would be advantageous. The direction of reconfigurable computing for the near future will be outlined, especially with processing acoustic signals in an array of distributed sensors.

Form SF298 Citation Data

Report Date *("DD MON YYYY")* 00102000	**Report Type** N/A	**Dates Covered (from... to)** *("DD MON YYYY")*

Title and Subtitle Application of Reconfigurable Computing to Acoustic Sensors	**Contract or Grant Number**
	Program Element Number
Authors Filipov, Andree; Falco, Mark	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) U.S. Army Research Laboratory Adelphi, MD 20783	**Performing Organization Number(s)**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Monitoring Agency Acronym**
	Monitoring Agency Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Document Classification unclassified	**Classification of SF298** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** unlimited
Number of Pages 15	

Introduction

There is a growing interest, especially in the U.S. Armed Forces, in the development of microsensor systems that are easily deployed and that support reconnaissance, surveillance, and target acquisition operations [1]. Such systems contain sensors, signal conditioning and processing subsystems, a radio link, and a power source. The role of microsensors is to autonomously *detect*, *classify*, and *localize* targets of interest in a variety of environments. It is generally thought that a number of different sensor types (acoustic, seismic, magnetic, and imaging) may be used to provide orthogonal features to aid in this detection, classification and localization. It is also desired that these systems be small (hand-carried, fit in a pocket), light (100s of grams), inexpensive (less than $200), easily deployable, and have a long operating life.

In most instances, power consumption (and therefore system lifespan) is a key consideration. Communication bandwidth is often limited by such power constraints and so the amount of data that can be transmitted is minimal. In addition, limiting the amount of data transmissions helps the microsensor avoid detection in instances where that is a concern. Communication is the largest consumer of battery power; therefore, much of the signal processing needed to implement the desired functionality must be performed within the previously mentioned size, power, and weight constraints of the microsensor itself to limit the amount of raw data transmitted and to maximize the amount of information. While these requirements highly constrain the microsensor components chosen, at the same time, they demand extremely high computational performance. In addition, as the field of microsensors matures, these demands will escalate. Finally, flexibility of the processing fabric is crucial to implement emerging algorithms and support a wide range of sensors and scenarios.

Candidates for microsensor processing include application-specific integrated circuit (ASIC) technology, general purpose processors (GPPs), and field-programmable gate arrays (FPGAs). Although ASIC technology can address the performance, power, and cost issues (assuming volume production levels), an ASIC solution is fixed, requiring a unique design for each sensor type or a large ASIC that can support a finite number of preexisting sensors. This poses a number of problems. The first issue is the nonrecurring engineering (NRE) costs associated with each ASIC development cycle. A large production run, not typical of military systems, is required to make this a cost-effective solution. This is exacerbated by ASIC development not lending itself to changes late in the design cycles. Errors or changes in the algorithm(s), as well as unforeseen environmental impacts on the algorithm subsequent to deployment (a major concern), will require the design to be respun. Each respin will incur additional NRE and development time. The second issue is that when an ASIC is developed, it is locked into the current silicon processing technology (number of usable gates, power/gate/frequency, operating voltage, etc). The design usually cannot then benefit from subsequent silicon process advancements unless it is converted or redesigned for a new process.

GPPs, which include both conventional CPUs and digital signal processors (DSPs), are more attractive than ASICs, in this application area, for a number of reasons. First, they provide flexibility to make changes, even after deployment. Second, they are driven by the commercial market and readily take advantage of new semiconductor processes. Third, they are standard parts and thus are marketed in large volumes, keeping costs of parts down. However, the most popular processors (Pentiums, PowerPC, etc) are not necessarily in the required power regime (watts vs milliwatts or less) and are not well suited for DSP. To address some of these shortcomings, some companies (Analog Devices, TI, etc) offer a number of DSP-specific processors that do provide the necessary signal processing performance, albeit at a relatively high power level. Additionally, there are also a number of GPPs designed for low-power embedded systems (StrongArm, MIPS, Motorola 56xxx, etc).

The performance of any GPP is limited by the computational resources it provides (ALUs, multipliers, etc). A performance price is often paid since the architecture is not tailored for the specific application for which it is being employed. Of interest to microsensors, this performance limitation may reveal itself as increased power because multiple devices would be necessary to meet the performance requirements. Finally, in a GPP, power cannot be reduced by controlling the number of gates or the number of gates switching simultaneously.

In small to moderate production quantities, typical of many military systems, FPGAs are almost always more cost effective than ASICs. In fact, recent literature suggests that even in volume, the cost per gate advantage that ASICs have over FPGAs is diminishing [2]. This reference goes on to say that "if the devices are (I/O) pad-limited, where the number of pads determines the die area, the costs for a programmable device and an ASIC are very close and may even favor the programmable device because of its higher production volume." Cost being equal, the FPGA would have the important benefit of being reprogrammable.

Common Architecture for Microsensors—A Reconfigurable Computing Architecture for Microsensors

The development of a standard architecture that supports a wide range of applications would reduce unit costs through higher volumes than are found in today's custom microsensor systems. In this section, we describe such an architecture.

Experience has shown that FPGAs are well suited to the development of deeply pipelined datapath applications typical in signal processing, while GPPs are better suited to system control and communication applications. The architecture we have developed (see Figure 1) allows the FPGA to act as a reprogrammable preprocessor or coprocessor to a GPP. The FPGA does most of the computationally complex pieces of the algorithm while the GPP is lightly loaded, performing control, communications, and housekeeping tasks. A by-product of this is that a simple low-power GPP can be used.

A goal of this work was to develop a flexible architecture that supports a wide range of low-power applications while still being able to leverage emerging commercial technology. Requirements for this architecture were further determined by reviewing the following Sanders microsensor projects: Defense Advanced Research Projects Agency's (DARPA's) Microair Vehicle (MAV), Microinternetted Unattended Ground Sensors (MIUGS), Small Unit Operations (SUO), U.S. Army Research Laboratory's (ARL's) Distributed Unattended NEtwork of Sensors (DUNES), Joint ARL and Communications Electronics Command (CECOM) science and technology objective (STO) for Warrior Enhanced Battlespace Sensors (WEBS), and Office of Naval Research's (ONR's) Autonomous Drifting Sensor (ADS). Through this effort, we determined that the basic processing flow, as depicted in Figure 2, was essentially the same for all of the cited applications.

This commonality in processing requirements led to the development of the Common Architecture for Microsensors (CAμS), pronounced "cause." Figure 3 shows a block diagram of the CAμS processing architecture consisting of a data acquisition module, DSP module, and FPGA module. CAμS is independent of a specific I/O technology and provides a large amount of user-defined digital I/O for interconnection to a variety of devices. Although we include a radio, the architecture can easily accommodate different radio technologies. The software and hardware can be programmed (configured) through a serial port with support to change the nonvolatile memory in situ. This nonvolatile memory is able to store four separate system configurations (DSP and FPGA configurations). Future work will provide the capability to remotely configure the system.

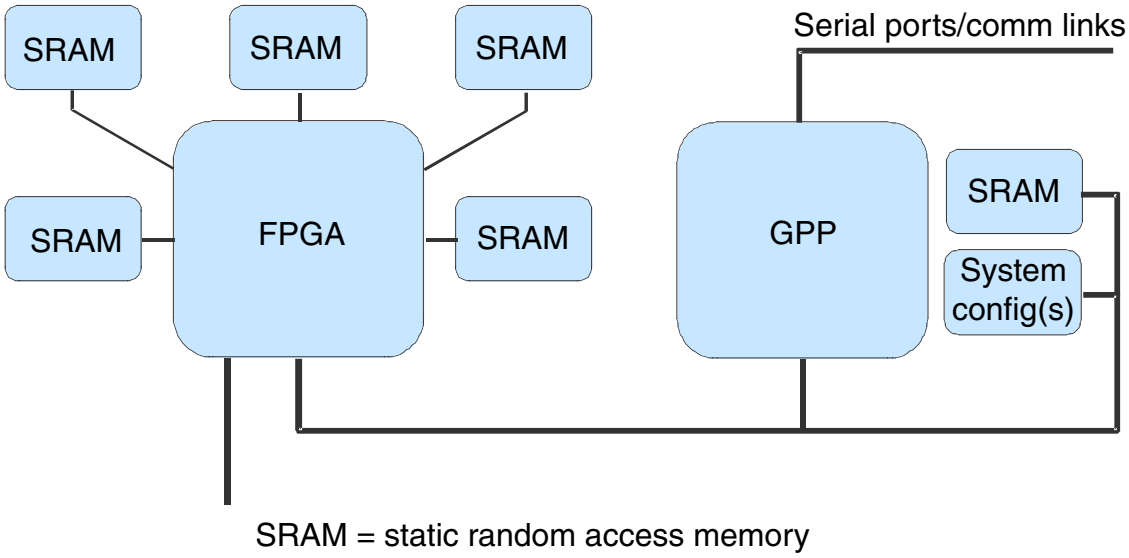
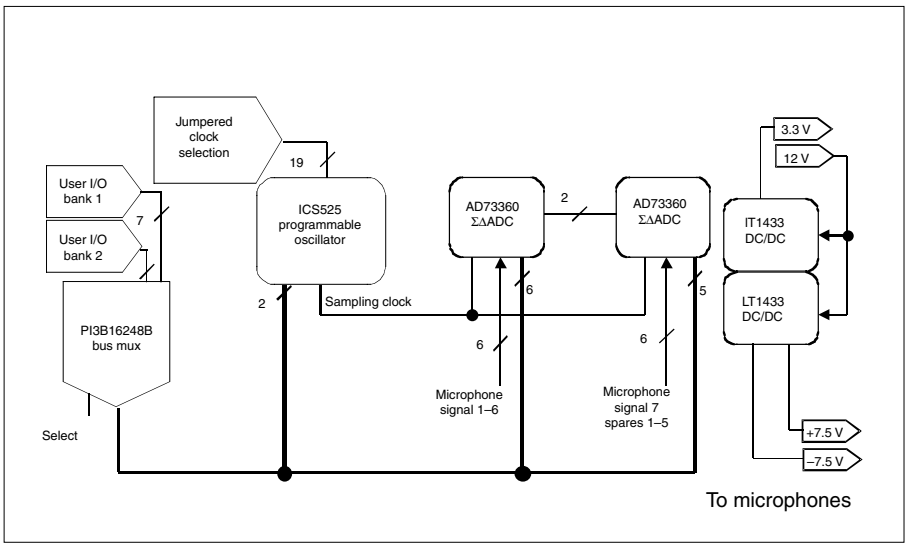


Figure 1. FPGA-DSP Combination.

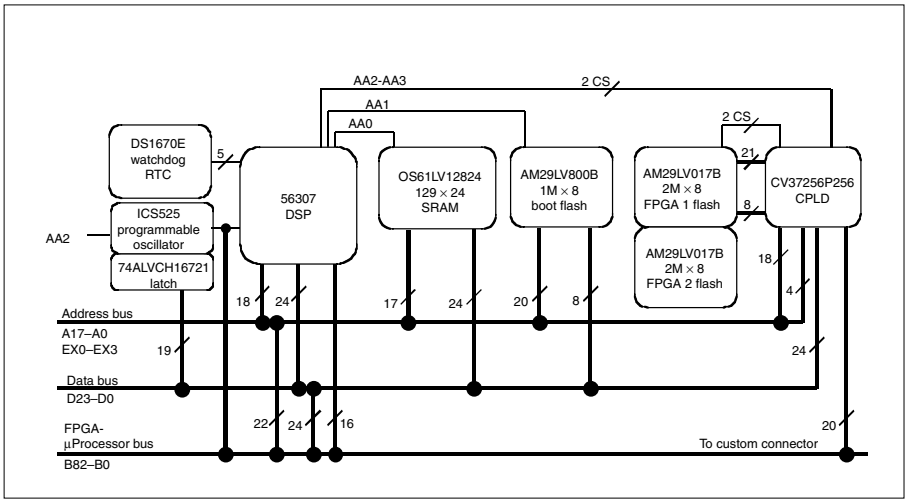
Figure 2. Common Processing Flow.

Figure 3. Block Diagrams of CAμS Modules, (a) CAμS data acquisition module, (b) CAμS DSP module, and (c) CAμS FPGA module.

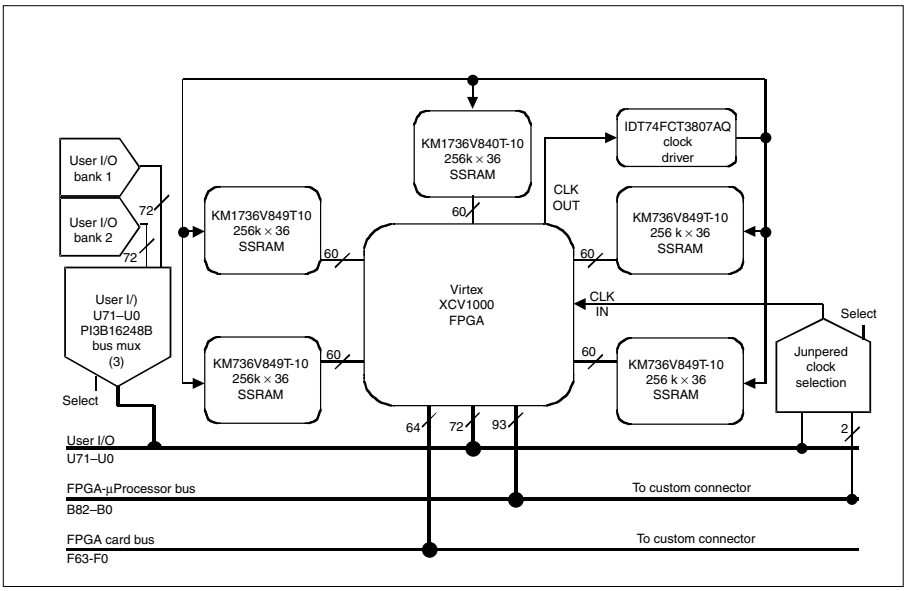
(a)



(b)



(c)



As depicted in Figure 2, the system is partitioned into sensor head, signal conditioning and processing, communication, and battery. An obvious system configuration is to integrate the processing, communication, and battery into a base unit, and the user could then attach sensor heads as needed for the application. Such a system could even act as a gateway for or a manager of a distributed sensor network. Our proof-of-concept system is based on a commercially available DSP and FPGA (the Motorola 56307 and the Xilinx Virtex XCV1000) in a standard PC-104 form factor (4-in. $\times$ 4-in. stackable cards). The system consists of either four or five cards: the DSP card, a power card, a data acquisition card, and one or two FPGA cards, depending on the application at hand.

As shown in Figure 3, the FPGA module consists of an FPGA and five static RAM memories of size 256 k $\times$ 36 bits each. Connections between the FPGA and the other boards allow the FPGA to control the data acquisition module and to interrupt the DSP. Additionally, the FPGA is connected to the DSP bus, making it possible to memory map FPGA resources into the DSP address space.

An Example CAμS Application—An Acoustic Sensor System for Target Detection and Bearing Estimation

The first CAμS application developed is an acoustic sensor system for ground-vehicle detection and bearing estimation. We describe this application here to show the trade-offs present in such a system and to illustrate the relative roles the GPP and FPGA can play. Note that this is simply an example; the range of applications and scenarios for which the technology is suited is tremendous.

The sensor array is formed by a collection of omnidirectional microphones, each providing 360° field-of-view coverage. The sensors are placed on or near the ground in a certain geometric shape to conform the array. The microphone array designed for application consists of seven microphones, six in an 8-ft-diameter circle and one in the center (Figure 4).

The acoustic detection application is shown in Figure 5. First, the signals received at each of the microphones are sampled at 1 kHz and the resulting data are multiplied by a windowing function. After a corner turn of the data, seven 1024-point fast Fourier transforms (FFTs) are performed. A detection algorithm is then run on the FFT output magnitudes to identify possible targets. This is done in two stages. First, a peak picking algorithm estimates the noise floor and high-power frequency bins are selected with a threshold determined from that noise floor. Harmonic line analysis is performed to find groups of frequency bins that form a harmonic relationship. Groups containing enough frequency components are declared to be targets. In the second stage, beamforming is performed on the raw data at the target's frequencies to determine exact target bearings. Finally, a tracking filter is applied to reduce the number of false reports and provide a lock on the target's bearing, providing lines-of-bearing (LOB) updates once every second.

Partitioning the Application on CAμS

This entire computation could be done on either the CAμS DSP or the CAμS FPGA alone. An evaluation of the characteristics of each subcomputation because of power consumption and throughput requirements, however, argues for a partitioning of the problem across both the DSP and FPGA.

*The XCV1000 FPGA has no standby mode. When the FPGA is powered on, its minimum power consumption is 200 mW. Source: Xilinx Virtex Power Estimator Spreadsheet, version 1.5.

Figure 4. Circular Acoustic Sensor Array.

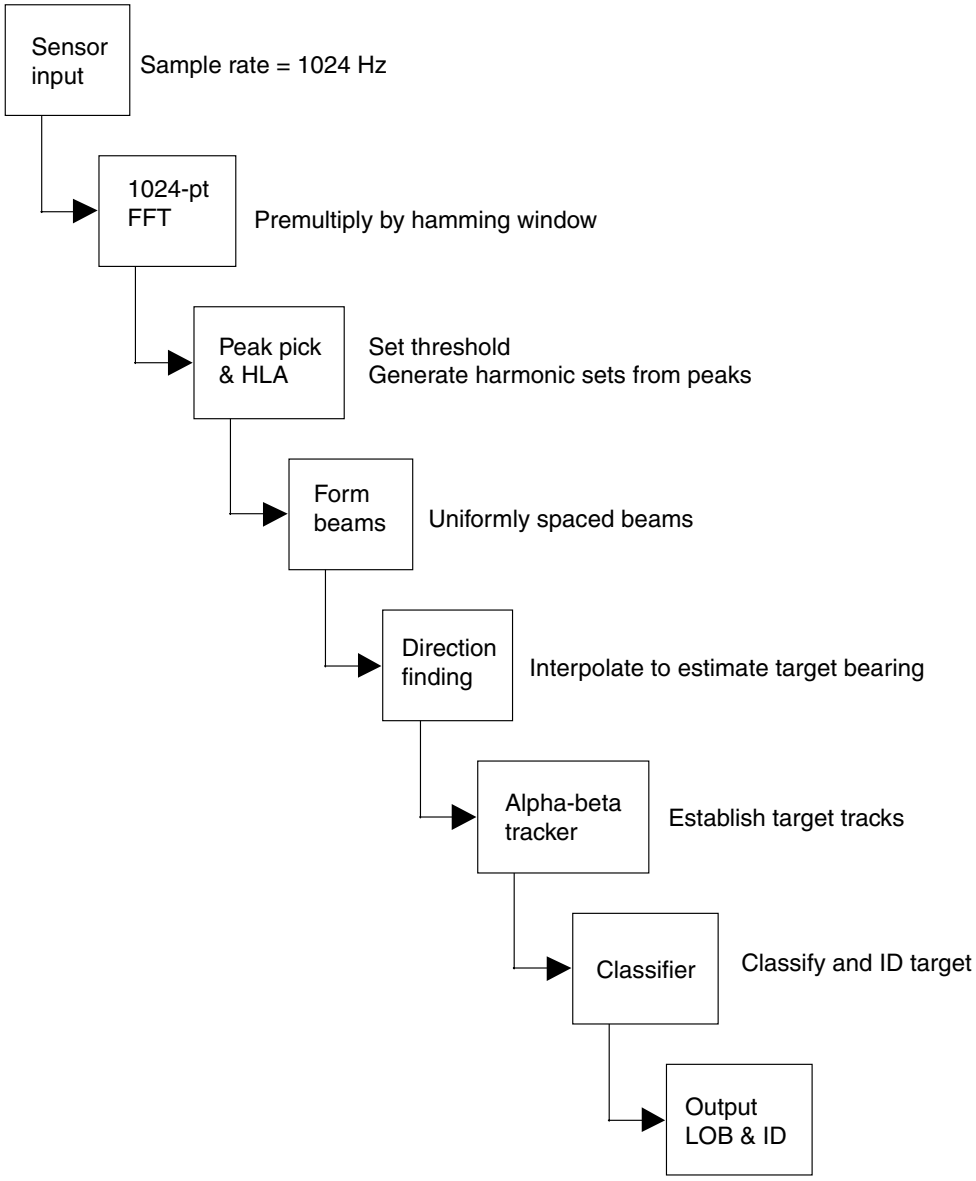


Figure 5. Signal Processing Flow Diagram.

The main consideration is power. The Motorola 56307 DSP and associated static random access memory (SRAM) consume approximately 40 mW when in standby mode, 510 mW at 256 kHz, and 1500 mW at 30 MHz. In contrast, an Xilinx XCV1000 FPGA has a quiescent power of 200 mW,* with additional power being a function of the design's gate count, operating frequency, and number of onboard memories used by the FPGA. Three options to use these two modules for this computation exist: (1) eliminate the FPGA, (2) use both the DSP and FPGA, and (3) eliminate the DSP.

For option (1), we estimate that the DSP module could do the whole computation with a clock rate between 256 kHz and 1 MHz. At this clock rate, its power consumption would be approximately 500 mW. For option (2), an attractive strategy would be to use the DSP module for only what it is best suited, run it at its highest clock rate, and keep it in standby mode as much as possible. The FPGA would then be used for the remainder of the computation and to minimize power and would be run just fast enough to meet the application's requirements. If the DSP module were run at 30 MHz and did only the minimum work required for detection, tracking, and communications, its power consumption would be approximately 43 mW (and in standby mode the most of the time). The FPGA module, running between 100 and 256 kHz, would consume at the most 225 mW for a total DSP + FPGA power consumption of 268 mW. This is about a 2 $\times$ reduction in power consumption over the DSP-only option. Finally, option (3) would use the FPGA for the full computation and eliminate the DSP entirely. We believe this is possible but would greatly complicate the application development for very little power reduction. The availability of the DSP greatly simplifies the FPGA design.

In its final form, this application was implemented via option (2) and consumed a total of 482 mW, including all system modules except the radio. The DSP was run at a high clock rate as just described, and the FPGA clock rate was 256 kHz (dictated by the A/D converter rate, since it and the FPGA module are closely coupled in the design). From this, one can see that eliminating the DSP and its 43 mW of power as in option (3) would have given a 9-percent overall power reduction at best.

This example shows that microsensor applications are an area where FPGAs can be used not only to increase system capabilities but also to significantly reduce power consumption. The development of low-power FPGAs would be extremely beneficial in this application area—note that the FPGA's quiescent power was almost half of the total system power and more than 90 percent of the FPGA power consumed. However, we need to emphasize that in this example, the computational demands were relatively light. With option (1), the DSP could have done the complete computation at a reasonably low clock rate, albeit at a much higher power consumption than our option (2) solution. Nevertheless, we believe that the power savings because of including the FPGA module may be even more pronounced in scenarios where the processing demands are higher as when beamforming at many different frequencies and across a larger sensor array. In this case, multiple DSP modules would be required to match the processing possible with a single FPGA [3] and the power savings would be more dramatic. However, determining the range of scenarios where this applies is a topic for future work.

Computing on the XCV1000

The flow diagram of the computation on the Xilinx XCV1000 FPGA is shown in Figure 6. The block diagram of the its design is shown in Figure 7. The incoming data format is 16-bit MSB-first bit-serial. The windowing unit does an MSB-first multiplication with Hamming coefficients stored in an on-chip table and converts the result to a 16-bit parallel word. The FFT unit does a 1024-point 24-bit FFT computation in just over 5100 cycles with a radix-2 constant-geometry FFT algorithm. The magnitudes of the FFT outputs are then stored in the DSP interface unit in BlockRam until needed. This dual-ported BlockRam is memory-mapped into the DSP's address space and provides a location for asynchronous data exchange between the

FPGA and DSP. FFT magnitudes are stored here for the DSP to use in its detection algorithm, and frequencies of interest are written in the BlockRam by the DSP for use in the FPGA's beamforming. LOB estimates computed by the beamformer are also written here for retrieval by the DSP.

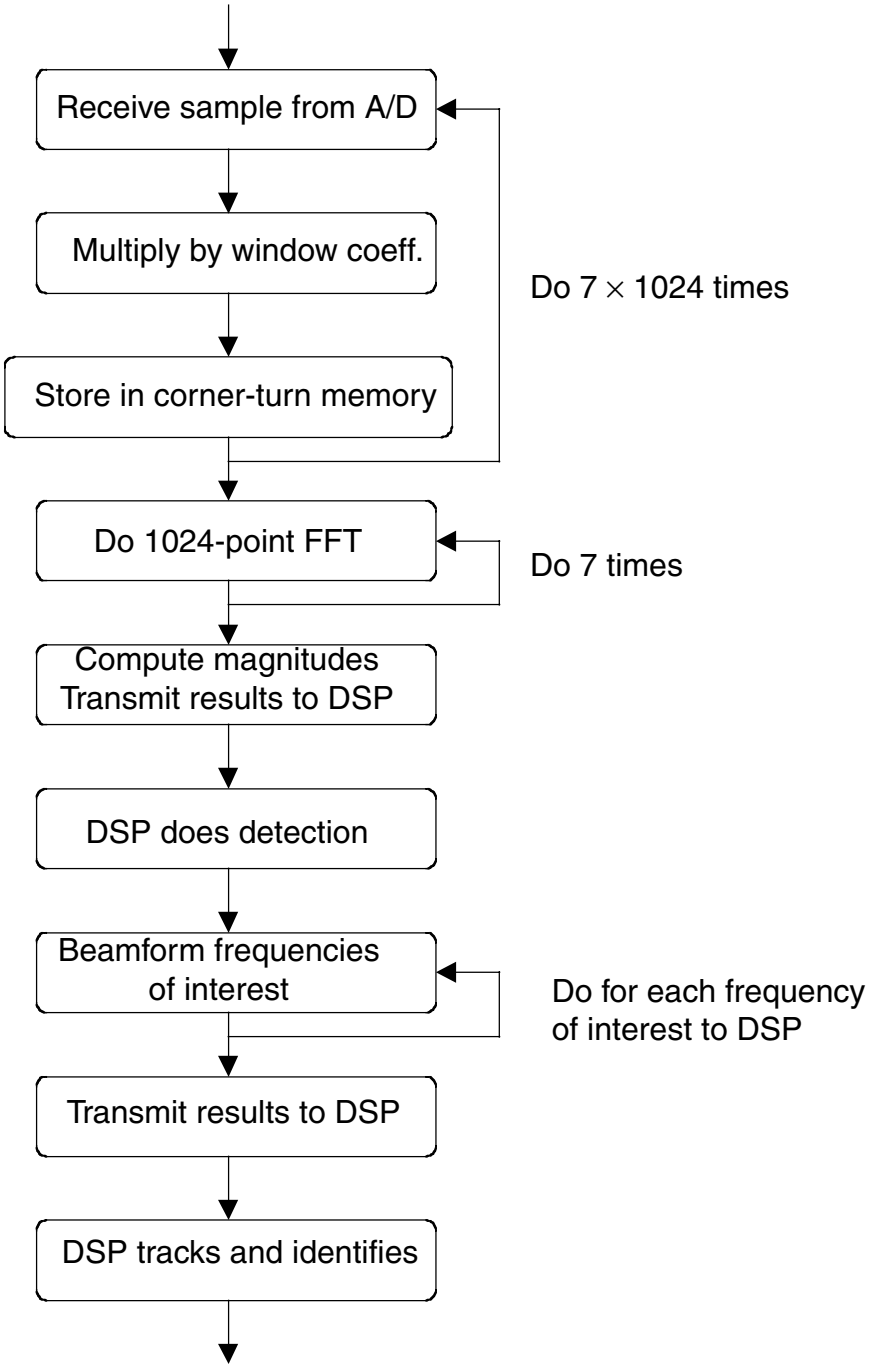


Figure 6. Flow Diagram of Computation.

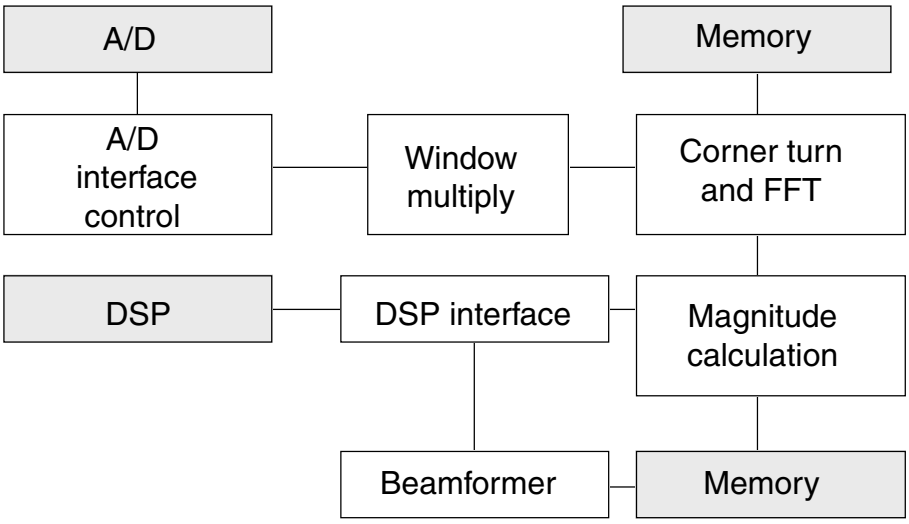


Figure 7. FPGA Module Computational Block Diagram (Shaded boxes are external to the FPGA).

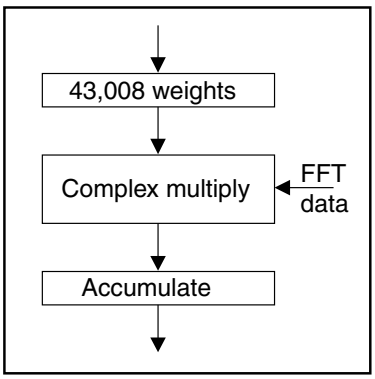
The FPGA design uses a number of circuit techniques to achieve low power. First, the ZBT memories on the board have a low-power sleep mode, which was used to reduce power. Second, in contrast to conventional high-performance designs, pipelining was not used in the datapath sections of the design. Rather, the datapath is combinatorial as far as is possible. Operand isolation, similar to that found in Correale's paper [4], was used to admit data to the combinatorial datapath only when a computation was desired. However, further implementation of such power-saving methods will be of limited usefulness given the quiescent power of the FPGA chosen.

A number of design features were incorporated into the beamformer to reduce its circuit area and computation time. Beamforming for a single frequency is described by the following pseudocode:

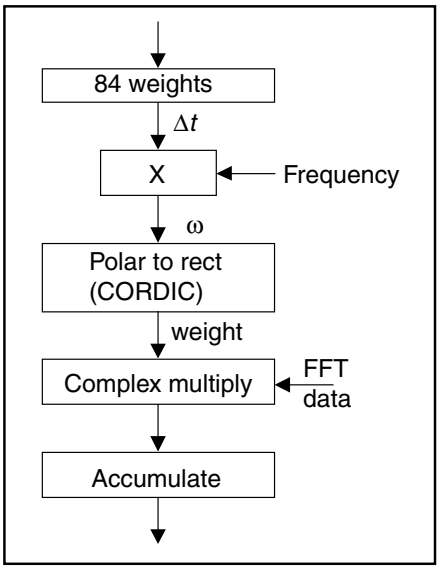
```
// f is frequency to beamform to
beamform(int f) {

 int max = 0; // max mag found so far
 int maxDir = 0; // direction of max

 for (d=0;d<NUMDIRECTIONS;d++) {
 accum = 0;
 for (s=0;s<NUMSENSORS;s++)

 // complex MAC
 accum += weights[d][f][s] *
 fftData[f][s];

 if (magnitudeOf(accum) > max) {
 max = magnitudeOf(accum);
 maxDir = d;
 }
 }

 // parabolic interpolation
 return interpolateDirection(max, maxDir);
}
```

Figure 8. Core FPGA Computations: (a) Original, (b) Memory Modification, and (c) Pipeline Modification (rect = rectangular).

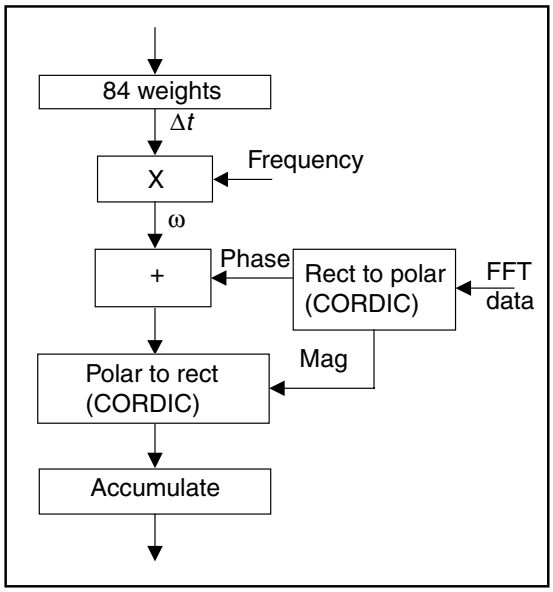
(a)



(b)



(c)



As can be seen in the code, the core computation is a complex dot product between a weight vector and a vector extracted from the FFT results. The largest such dot product result determines the direction of the source. In this application, the number of frequencies is 512, the number of directions is 12, and the number of sensors is 7. A straightforward implementation of this computation would thus require 43,008 complex weights to be stored in an additional memory and a complex multiply-accumulate operation as shown in Figure 8(a).

However, each weight has the form of a complex exponential: $e^{j\Omega}$ that can also be expressed as $e^{\{j*f*\Delta t\}}$. In our design, we exploit this and only store the Δt values required (as in time-delay beamforming). The required phase rotation to apply can then be determined via a multiplication by the frequency of interest. This reduces the memory required from 43,008 complex weights (172 kB) to 84 Δt values (168 B), making storage on chip possible.

This optimization reduces the memory required but now requires a polar-to-rectangular conversion (Figure 8(b)). Our final design avoids the complex multiply completely. As shown in Figure 8(c), if we first convert the FFT data to polar form, we can simply add a phase rotation to the FFT phase term. A final conversion back to rectangular coordinates makes the desired accumulating result possible.

A CORDIC unit is used for both conversions. In an FPGA, an unrolled CORDIC unit takes about the same area as a multiplier. The end result is that we have replaced an off-chip memory and a complex multiply by two CORDICs and thereby obtained a large savings in circuit area and power consumption. In addition, we used CORDIC for all magnitude calculations elsewhere in our design, further reducing circuit area and power. This extensive use of CORDIC throughout accounted for much of the computational advantage the FPGA had over the DSP. However, while the method shown in Figures 8(a) and 8(b) could be used to save memory, the CORDIC computations shown in Figures 8(b) and 8(c) would *not* be executed efficiently on the DSP. A more complete description of this and related techniques as applied to sonar beamforming on FPGAs is given in a paper by Nelson [3].

Results

In the end, this CAμS design was extremely successful—demonstrating a dramatic reduction (more than *four orders of magnitude*) in size*weight*power over the existing ARL's DUNES acoustic testbed using a full Compact-PCI chassis. These improvements were achieved without any sacrifice in performance as is shown by a typical data run through both the CAμS (Figure 9), and the ARL testbed (Figure 10). This result is the first demonstration of the tremendous benefits and potential for using FPGA based processors for distributed microsensors.

A number of factors are combining to make FPGAs a technology enabler for future microsensor systems. Limited communications bandwidth and the high power associated with communications are forcing the migration of signal processing to the sensor head. The superior performance/power characteristics of FPGAs for many signal conditioning and processing algorithms, coupled with a desire to support unattended sensor lifespans of up to a year, points to FPGAs as the technology of choice for these computations. Figure 11 is a photograph of the CAμS hardware. It replaced a full Compact-PCI chassis for this application. Since, the implementation of the acoustic algorithm consumed only a part of the CAμS hardware resources, the capability exists to easily augment or modify the existing algorithm.

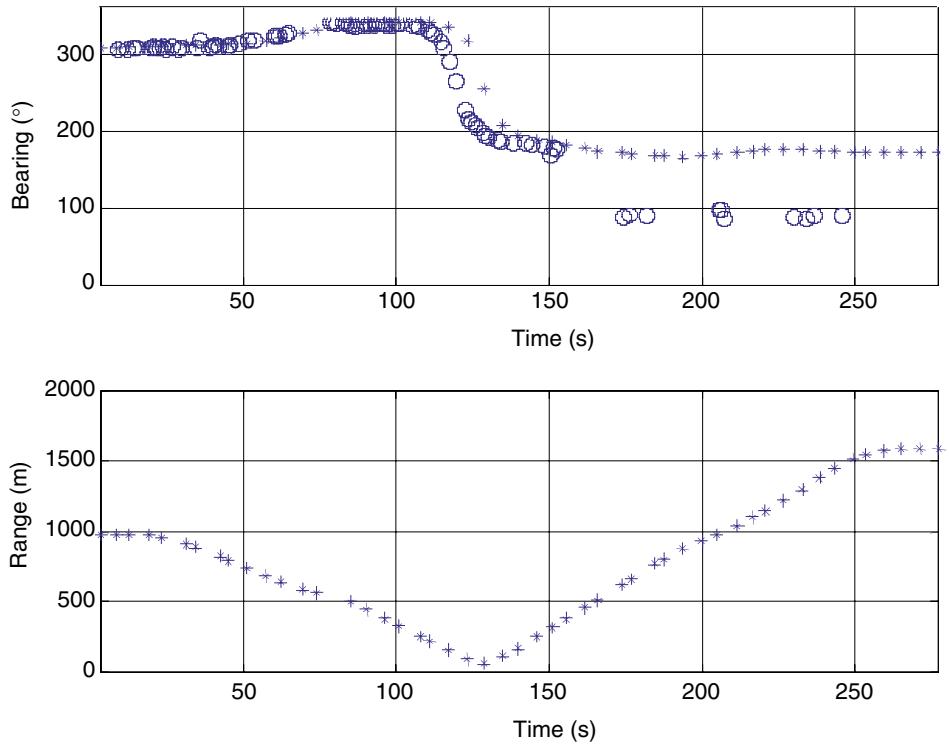


Figure 9. CAμS LOB versus Ground Truth for Typical Run: (a) R1_apg7_10_4, target ID @ Speed(kph): 1@48.276 and (b) Distance from Target to Sensor Array.

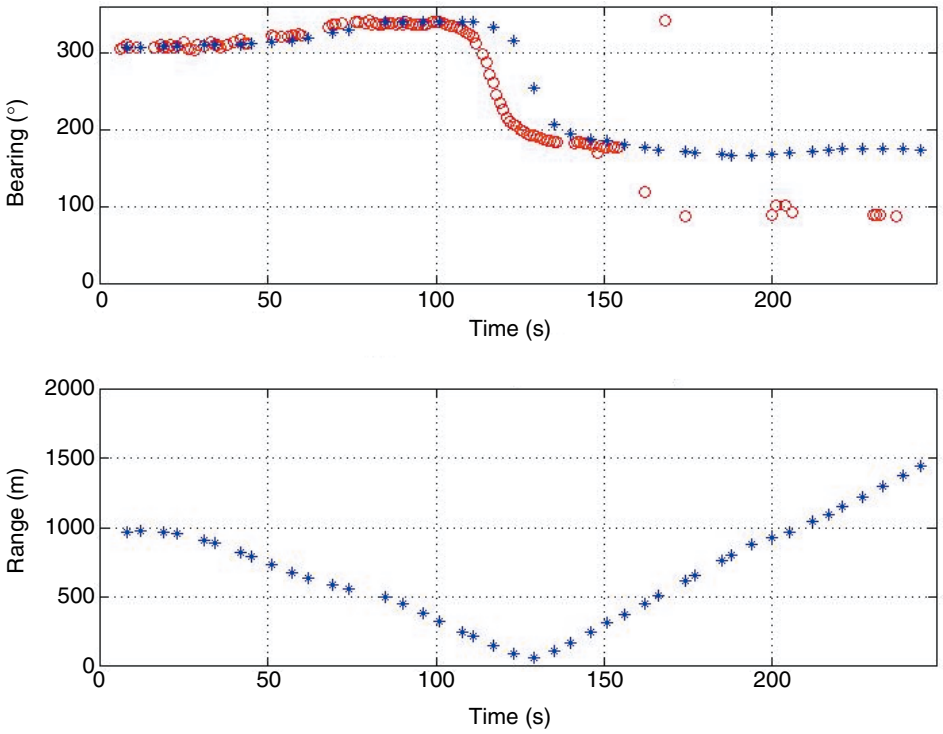
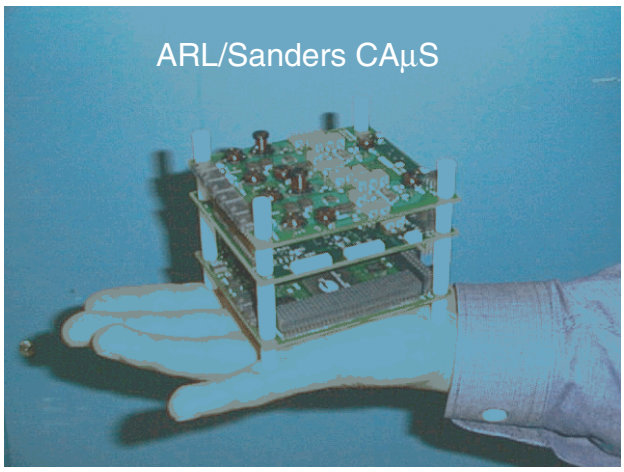


Figure 10. DUNES LOB versus Ground Truth for Typical Run: (a) R1_apg7_10_4, target ID @ Speed(kph): 1 @ 48.276 and (b) Distance from Target to Sensor Array.

Figure 11. CAμS Hardware.



Future Work

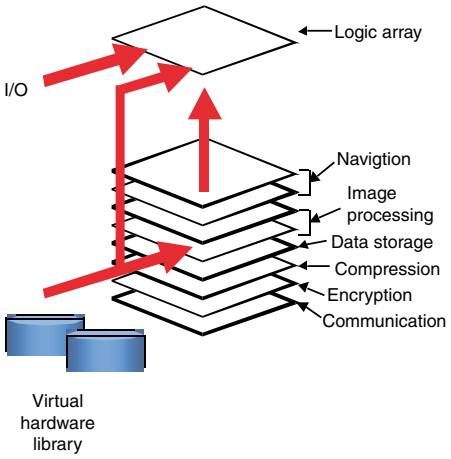
Future work will focus on adapting CAμS for use with additional sensors and on miniaturizing the CAμS architecture. In this upcoming microsensor system, the FPGA-based processor will be capable of supporting magnetic, acoustic, seismic, and imaging (day or infrared (IR)) sensors all in the same package. The FPGA will have to be reconfigured in real time to facilitate such a sensor suite. In a microsensor environment, gaps in the data collected while reconfiguration is accomplished are often acceptable—the targets being monitored (people and vehicles) are relatively slow-moving. For example, the FPGA can run the acoustic detection/LOB estimation algorithm described in this paper to cue an IR camera. The FPGA could then be reconfigured to run an automatic target recognition (ATR) algorithm to chip out regions of interest (ROIs). Additional reconfigurations would be used to compress the image chips and transmit them back to an operator. The best of today's commercially available technologies requires several milliseconds to configure. There are many real-time situations where this reconfiguration time is prohibitive. To accommodate these situations, Sanders has developed a dynamically reconfigurable FPGA that can reconfigure in a single clock cycle [5].

The context-switching reconfigurable computing (CSRC) technology now being developed by Sanders extends commercially available FPGA devices, to include high-speed changes between a number of programmed functions without the need for additional FPGAs. Each configuration, referred to as a *context,* in a CSRC FPGA has the functionality similar to that of many commercially available FPGAs. The context switching can occur at significantly higher speeds than the rate at which current FPGA technology can reconfigure. In addition, unlike commercial FPGAs, where reprogramming destroys any resident data, the CSRC FPGA can share data between contexts.

The concept of virtual hardware is an obvious benefit of dynamic reconfiguration. If configurations can be swapped in and out of an FPGA upon demand at a real-time system rate, only the necessary hardware needs be instantiated at any given time. In this manner, a virtually infinite algorithm cache or an infinite coprocessor can be conceived. In other words, a high-level system scheduler can instantiate hardware as needed. In this manner, a reduction in size, weight, and power can be achieved. Additionally, given the CSRC FPGA, if the processing requirements specify a sequential application of algorithms, the context layers can be set up to share data so that the output of one algorithm is immediately available as the input to the next algorithm upon a context switch. This is not possible with contemporary FPGAs.

A natural extension of the algorithm cache mode of computation is the concept of mission-phase reprogrammability. As seen in Figure 12, an entire mission can be mapped to a CSRC device. In this case,

Figure 12. Reconfiguration Benefits.



- Mission Phase Reconfiguration
  - –Navigation
  - –Destination recognition
  - –Image processing: Target recognition
  - –Data compression
  - –Data encryption
  - –Radio waveform generation
  - –Transient data storage
- Data Dependent Reconfiguration
  - –Image classification template match
  - –Threshold-sensitive filter selection
- Software Acceleration
  - –Dynamic link libraries (DLL) in hardware called by software
- Virtual Hardware
  - –Stored "library" of configurations
  - –Hardware caching
  - –Configuration available and configuration memory shift over time

different contexts can house different algorithmic phases of a mission without requiring an algorithm to be confined to a single context, depicted as layers in Figure 12.

Although Figure 12 identifies the obvious modes of computation for gaining a performance enhancement, it is believed that the true potential of context switching requires a paradigm shift in algorithm implementation. The capabilities of the CSRC architecture, which extend dynamic reconfiguration to context switching, can provide improved implementations of signal processing algorithms over those currently available through commercial FPGAs. The inherent ability of CSRC to quickly perform different tasks and share results among different configurations allows one to approach algorithms from a different perspective, enabling mathematical implementations previously inconceivable without context switching.

As noted earlier, FPGAs are still quite power hungry. Under Federation Laboratory (FedLab) funding, the Massachusetts Institute of Technology (MIT) has developed a very low-power programmable DSP that achieved almost six orders of magnitude power reduction over a Strong Arm processor.* This has been achieved by going to a 0.9-V process, through clever architecture to facilitate computing with the use of the optimal number of bits, and by conditional clocking, to highlight the primary techniques. MIT is now proposing to apply this same design methodology to the Sanders-designed CSRC[5] processor. If this is successful, then any objections to the high-power nature of FPGAs will be removed and reconfigurable computing will become the clear choice for microsensor applications.

References

1. Shinseki and Caldera, "The Army Vision: Soldiers On Point for the Nation … Persuasive in Peace, Invincible in War," *The Assistant Secretary of the Army Acquisitions Logistics Technology*.
2. Tets Maniwa, "Focus Report: Programmable Logic," *Integrated System Design*, October 1999, pp 42–50.
3. P. Graham and B. Nelson, "FPGA-Based Sonar Processing," *Proc. of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, February 1998, pp 201–208.
4. A. Correale, Jr., "Overview of the Power Minimization Techniques Employed in the IBM PowerPC 4xx Embedded Processors," *Proc. International Symposium of Low Power Design*, April 1995, pp 75–80.
5. Sanders, "The Design and Implementation of a Context Switching FPGA," *IEEE Symposium on FPGA's for Custom Computing Machines*, 15–17 April 1988.

*Demonstration of MIT Low-Power Processor—FedLab Symposium.